Методичка 7.4 - Применение и обход антиотладочных приемов. Часть 3.

Принцип работы упаковщиков

В первом видео этого урока мы уже немного говорили про упаковщики. Основная цель использования упаковщиков заключается в том, чтобы скрыть код оригинальной программы от статического анализа. Статический анализ, на самом деле, выполняют не только люди, но и другие программы, например, антивирусы. Возможно, вы слышали такой термин, как сигнатурный анализ. Антивирусы проверяют файлы на предмет наличия сигнатур различного вредоносного ПО, чтобы понять, опасен этот файл или нет. Упаковщики позволяют скрыть код программы от такого анализа и обойти подобную проверку.

Для того, чтобы скрыть оригинальный код программы, он может быть зашифрован с использованием любого алгоритма шифрования. После шифрования код программы превращается в обычный блок данных, который ничего общего с кодом не имеет. Этот блок данных можно назвать шифр-текстом.

После того, как шифр-текст получен, упаковщик добавляет к нему специальный код (распаковщик), который будет расшифровывать шифр-текст, а также новый PE-заголовок, где точка входа (entry point) будет указывать на распаковщик.

При запуске такого файла распаковщик выполняет распаковку зашифрованного блока данных куда-то в память и передает на них управление. Все эти операции происходят только в оперативной памяти, на жестком диске оригинальный код программы не появляется.

Если мы попробуем выполнить статический анализ такой программы, то увидим только какой-то блок данных и код распаковщика, который что-то делает с этим блоком данных.

Для того, чтобы получить оригинальный код программы необходимо запустить запакованную программу под отладчиком. Позволить распаковщику выполнить расшифровку блока данных и в этот момент приостановить выполнение программы. Далее мы можем либо сделать дамп оперативной памяти, либо выполнить анализ оригинального кода программы прямо в отладчике.

Применение упаковщика UPX

Для того, чтобы продемонстрировать процесс упаковки бинарного файла мы воспользуемся бесплатным упаковщиком UPX.

Рекомендации по установке упаковщика UPX

Ссылка: https://github.com/upx/upx/releases

Будем упаковывать программу prog-5.4.c

Исходный код программы prog-5.4.c

```
#include <stdio.h>

int main(int argc, char* argv[]) {
    printf("Hello, World!\n");
    return 0;
}
```

Компилируем программу

```
> cl prog-5.4.c -Od /Gs- /GS- /link /DYNAMICBASE:NO /NXCOMPAT:NO
```

Запускаем.

```
> prog-5.5.exe
Hello, World!
```

Попробуем упаковать программу.

```
> upx.exe prog-5.4.exe -o prog-5.4-packed.exe

...
96256 -> 52224
...
```

Видим, что размер программы уменьшился почти в 2 раза. Стоит отметить, что упаковщики используются не только для того, чтобы затруднить статический анализ, но и для уменьшения размера бинарного файла.

Давайте попробуем открыть в отладчике OllyDbg упакованную программу prog-5.4-packed.exe

Отладчик OllyDbg предупреждает нас о том, что файл упакован и проанализировать его будет не просто.

Видим, что перед нами инструкции, которые точно не похожи на код нашей программы. Закрываем отладчик.

При этом упакованная программа работает так же как и оригинальная.

```
> prog-5.4-packed.exe
Hello, World!
```

Давайте запустим утилиту PEEE и изучим запакованный бинарный файл.

Переходим в раздел Section Header и видим, что в файле 3 секции: UPX0, UPX1 и UPX2.

Давайте выясним, какая из них является секцией кода, а какая секцией данных.

Читаем PE-заголовок.

NT Header - Optional Header - BaseOfCode - 0x0000E000
NT Header - Optional Header - BaseOfData - 0x0001E000

Возвращаемся в раздел с секциями - Section Header и видим:

UPX0 - 00001000
UPX1 - 0000E000
UPX2 - 0001E000

Значит секция кода - это UPX1, а секция данных - UPX2.

Проверяем точку входа (entry point):

NT Header - Optional Header - AddressOfEntryPoint - UPX1.

Точка входа указывает на секцию UPX1, значит в секции UPX1 расположен распаковщик.

Исходя из описания в PE-заголовке секция UPX0 есть, но не используется. Это признак того, что секция UPX0 будет использоваться в служебных целях.

Распаковка бинарного файла

Давайте попробуем распаковать бинарный файл.

В нашем случае известно какой упаковщик был использован для нашей программы, но в обычной жизни разработчик скорее всего вам такой информации не даст.

Для определения упаковщика есть замечательная утилита - PEiD.

Рекомендации по установке утилиты PEiD

Ссылка: https://www.aldeid.com/wiki/PEiD

Result: UPX 0.89.6 - 1.02 / 1.05 - 2.90 -> Markus & Laszlo

Видим, что программа смогла определить упаковщик - это UPX.

Самый простой способ распаковать данный файл, это найти в сети Интернет распаковщик для пакера UPX, но такой способ работает только тогда, когда пакер широко известен.

Может случиться так, что файл будет запакован каким-нибудь самописным пакером и вы не сможете найти подходящего распаковщика в сети Интернет, поэтому мы попробуем распаковать бинарный файл самостоятельно в ручном режиме.

Запускаем отладчик, открываем запакованную программу prog-5.4-packed.exe.

Давайте посмотрим на наше адресное пространство - нажимаем кнопку M в интерфейсе отладчика OllyDbg.

PE-заголовок расположен по адресу - 0x00400000.

Также мы видим секцию UPX0 - 0x00401000. Давайте посмотрим, что в ней находится. Если дважды кликнуть по ней, то мы увидим ее содержимое. Сейчас там нулевые байты.

Следом за этой секцией идет секция кода UPX1 и секция данных UPX2. Давайте приступим к анализу бинарного файла.

Видим, что первая инструкция, на которой остановился отладчик - это PUSHAD.

Эта инструкция сохраняет все значения регистров и флагов в стеке. Есть обратная инструкция POPAD, которая восстанавливает значения регистров и флагов из стека.

Упаковщик использует эти инструкции, чтобы в момент передачи управления на распакованный код, все регистры были в таком же состоянии, как и при старте программы.

Наша задача заставить распаковщик распаковать код и остановить отладчик перед передачей управления на распакованный код.

Для того, чтобы нам поймать момент, когда упаковщик будет передавать управление на распакованный код, мы можем установить аппаратную точку останова на данные в стеке.

Как только будет обращение к этим данным, сработает аппаратная точка останова.

Ставим аппаратную точку останова.

Кликаем по области данных в отладчике OllyDbg и нажимаем Ctrl + G (перейти по адресу) и указываем в качестве адреса ESP - 4.

Это адрес с которого будут начинаться байты, которые попадут в стек по инструкции PUSHAD.

Устанавливаем аппаратную точку останова на первом байте. Кликаем правой кнопкой на первом байте - Breakpoint - Hardware, on access - Byte.

Посмотреть действующие аппаратные точки останова можно следующим образом.

Debug - Hardware Breakpoints

Давайте продолжим выполнение программы - F9.

Отладчик остановил выполнение программы сразу после инструкции POPAD. Это то, что нам нужно. Сейчас код нашей программы распакован, но мы пока точно не знаем, где он расположен в памяти.

Давайте уберем аппаратную точку останова, она нам больше не понадобится.

Debug - Hardware Breakpoints - Delete 1 - OK

Теперь посмотрим, какие данные расположены в секции UPX0 после того, как отработал распаковщик.

Кликаем по области кода в отладчике OllyDbg и нажимаем Ctrl + G (перейти по адресу) и указываем в качестве адреса адрес секции UPX0 (0x00401000).

Видим распакованный код нашей программы, теперь мы можем поставить точку останова в начале функции, нажать F9 и выполнить ее анализ.